Постановка задачи

Рассматриваем антагонистические матричные игры. Задача: поиск значения игры и оптимальных стратегий игроков.

Антагонистическая игра - некооперативная игра, в которой участвуют два игрока, выигрыши которых противоположны.

Антагонистическая игра задается совокупностью $\Gamma = \langle X, Y, F(x, y)\rangle$:

В игре принимают участие два игрока - первый и второй. Первый выбирает стратегию x из множества стратегий X, второй выбирает стратегию y из множества стратегий Y. Задана **функция выигрыша** F(x, y) первого игрока, определенная на (X × У), где (A × B) - декартово произведение множеств A и B. Выигрыш F(x, y) первого игрока является проигрышем для второго. Цель первого игрока состоит в увеличении своего выигрыша F(x, y), а цель второго — в уменьшении F(x, y).

Антагонистическая игра Г называется **матричной**, если множества стратегий игроков конечны: $X = \{l, ..., m\}$, $Y = \{1, ..., n\}$. При этом принято обозначать стратегию первого игрока через i, стратегию второго через j, а выигрыш первого F(i, j) через a_{ij}. Матрица A = a[i, j]; i = 1..m, j = 1..n называется матрицей игры. Первый игрок выбирает в ней номер строки i, а второй — номер столбца j.

Рассмотрим игру $\Gamma = \langle X, Y, F(x, y)\rangle$.

Пара (x^0, y^0) из (X × У) называется **седловой точкой** функции F(x, y) на (X × Y), если выполняется

$$F(x, y^0) \le F(x^0, y^0) \le F(x^0, y), \ \forall x \in X, \forall y \in Y$$

В обозначениях матричной игры (i^0, j^0) - седловая точка матрицы A, если

$$A[i, j^0] \le a[i^0, j^0]) \le a[i^0, j], \ i = 1..m, j = 1..n.$$

Говорят, что антагонистическая игра Г имеет **решение**, если функция F(x, y) имеет на (X × Y) седловую точку. Пусть (x^0, y^0) - седловая точка функции F(x, y). Тройка $(x^0, y^0, v = F(x^0, y^0))$ называется решением игры, пара x^0, y^0- **оптимальными стратегиями** игроков, а v — **значением** игры.

Смешанной стратегией первого игрока в игре Г называется вероятностное распределение P на множестве стратегий X.

Пусть $X = \{1, ..., m\}$, как это имеет место в матричной игре. Тогда вместо P для обозначения смешанной стратегии будем использовать **вероятностный вектор** p = (p[i], ..., p[m]), удовлетворяющий ограничениям

$$p[1] + ... + p[n] = 1, \quad p[i] \geq 0, i = 1..m.$$

Если применяется вектор p, то стратегия i выбирается с вероятностью p[i].

Обозначим через {P} - множество всех смешанных стратегий первого игрока на множестве X. Можно считать, что $X \subset \{P\}$. Если множество X конечно, то выбор i стратегии эквивалентен выбору смешанной стратегии p = (**0**, ..., **0**, **1**, **0**, ..., **0**), где единица стоит на i-м месте. Множество X будем называть множеством **чистых стратегий** первого игрока (в противовес смешанным).

Матричная игра Г задается матрицей A = (a[i, j]). В ней множество смешанных стратегий первого игрока —

$$\mathbf{P} = \{p= (p[1], ..., p[m]); p[1] + ... + p[m] = 1, p[i] \geq 0, i = 1..m.\}$$

множество смешанных стратегий второго игрока -

$$\mathbf{Q} = \{q= (q[1], ..., q[n]); q[1] + ... + q[n] = 1, q[j] \geq 0, j = 1.. n.\}$$

Решение (p, q, v = F(p, q)) игры Г называется решением исходной игры Г в смешанных стратегиях. При этом p, q называются оптимальными смешанными стратегиями игроков, а v — значением игры Г.

Всякая матричная игра имеет решение в смешанных стратегиях.

По условию нам дана матрица игры A = (a[i, j]) произвольного размера $m \times n$. По ней нужно найти значение игры v и оптимальные смешанные стратегии игроков p` и q` - первого и второго соответственно.

Математическое решение

Ищем значение игры и оптимальные стратегии игроков с помощью сведения решения матричной игры к паре двойственных задач линейного программирования (ЛП), которые затем решим двойственным симплекс-методом.

Задача ЛП в стандартной форме:

$$Z = c[1] \times x[1] + ... + c[n] \times x[n] \rightarrow \max$$

$$a[i, 1] \times x[1] + ... + a[i, n] \times x[n] \leq b[i], i = 1..m; \qquad (1)$$

$$x[j] \geq 0, j = 1..n; \qquad (2)$$

Z называется **целевой функцией,** которую следует максимизировать при линейных ограничениях (1), (2).

Если линейные ограничения заданы в форме равенств и выполняется (2), то говорят, что задача ЛП задана в **канонической форме**.

Опишем решение заданной задачи **симплекс-методом**:

Задачу в стандартной форме приводим к канонической форме, вводя **слабые** переменные x[n + 1], ..., x[n + m], получим линейные ограничения:

$$a[i, 1] \times x[1] + ... + a[i, n] \times x[n] + x[n + i] = b[i],\ i = 1..m;$$
$$x[j] \geq 0,\ j = 1..n;$$

Допустимое решение задачи в канонической форме — вектор значений (x[1], …, x[n + m]), удовлетворяющий ограничениям задачи. Множество допустимых решений — многогранник. Его вершины — **базисные допустимые решения** (БДР).

Допустимое решение является **БДР**, если столбцы матрицы, составленной из исходной матрицы ограничений присоединением справа единичной матрицы порядка m, отвечающие номерам i: x[i] != 0 из вектора допустимого решения линейно независимы.

Базис матрицы - система л.н.з. столбцов матрицы $\{A[j], j \in J\}$, J будем называть **множеством базисных номеров** (МБН).

При условии $b[i] \geq 0$, i = 1..m, как начальное БДР обозначим вектор x(1) = (0, …, 0, b[1], …, b[m])

Далее будем двигаться по ребрам многогранника, тем самым увеличивая целевую функцию.

Построим ребро многогранника, постепенно увеличивая от нуля переменную x[p], $p \notin J$. Для этого изменяем базисные компоненты, чтобы полученное решение оставалось допустимым, получим вектор

$$x = (0, ..,0, x[p], 0, .., 0, b[1] - a[1, p] \times x[p], ..., b[m] - a[m, p] \times x[p])$$

Это решение допустимо, если $b[i] - a[i, p] \times x[p] \geq 0$, i = 1..m;

Нас интересует случай когда есть i: a[i, p] > 0 (иначе не существует конечного оптимального решения). Тогда

$$0 \leq x[p] \leq \theta = b[l] / a[l, p]$$

$\theta = \min(b[i] / a[i, p])$ по всем i: a[i, p] > 0;

Решение x пробегает ребро, если $\theta > 0$, один конец ребра - решение x(1), другой - решение

$$x(2) = (0, .., 0, \theta, 0, .., 0, b[1] - a[1, p] \times \theta, ..., b[m] - a[m, p] \times \theta)$$

На позиции n+l получится 0. Получили новое БДР с МБН $J(2) = (J(1) \setminus \{n+l\}) \cup \{p\}$.

Здесь столбец A[p], строка с номером l и элемент матрицы a[l, p] называются ведущими.

Целевая функция на этом ребре $Z = c[p] \times x[p]$. Если $\theta > 0$ и c[p] > 0, то Z возрастает, поэтому номер p выбираем из условия c[p] > 0.

Выше мы описали переход от одного БДР к другому, этот алгоритм останавливается, если:

1) $a[i, p] \le 0$, $i = 1..m$; $c[p] > 0$. В этом случае оптимального решения нет.
2) $c[j] \le 0$, $j = 1..n$, тогда решение $x(1)$ является оптимальным.
Если $c[p] > 0$ и существует i: $a[i, p] > 0$, алгоритм продолжает работу.
Но перед следующей итерацией алгоритма следует эквивалентно преобразовать исходную задачу, так чтобы для решения $x(2)$ выполнялись те же свойства, что и для $x(1)$ перед первой итерацией симплекс-алгоритма. Эти свойства:
1) Базисные столбцы исходной задачи, соответствующие решению $x(1)$, образуют единичную матрицу.
2) Целевая функция Z зависит только от небазисных переменных.

Пусть $a[l, p]$ - ведущий элемент. Для выполнения свойства (1) нужно элементарными преобразованиями строк матрицы добиться того, чтобы столбец с номером p стал единичным столбцом с единицей на l-ой строке.

Для выполнения свойства (2) преобразуем Z следующим способом:

$$Z = (c[1] \times x[1] + ... + c[n] \times x[n]) - c[p] / a[l, p] \times ((a[l, 1] \times x[1] + ... + a[l, n] \times x[n]) + x[n + l] - b[l])$$

Таким образом, если решение x удовлетворяет ограничениям исходной задачи, то вычитаемое выражение обращается в 0 и Z не изменяется. После преобразования Z не зависит от базисных переменных $x(2)$, так как коэффициент при $x[p]$ равен нулю.

Запишем новую формулу Z:

$$Z = c`[1] \times x[1] + ... + c`[n + m] \times x[n + m] + c`[0]$$

где $c`[0] = (c[p] \times b[l]) / a[l, p] = c[p] \times \theta$; Z при подстановке $x(2)$ обратится в ноль.

Проделав подобные преобразования, можно от БДР $x(2)$ переходить к БДР $x(3)$ описанным выше способом. Повторять итерации до выполнения условий остановки алгоритма.
Таким образом, с помощью симплекс-метода получим оптимальное решение задачи ЛП (либо докажем, что такого нет).

Сведение к двойственной задаче

Теперь опишем сведение матричной игры к **паре задач ЛП.**

Без потери общности, положим, что значение игры $v > 0$. Значение игры представимо в виде:

$$v = \max (\min (A(p, j))) = \max(\min(p[1] \times a[1, j] + ... + p[m] \times a[m, j])$$

где max берется по всем смешанным стратегиям первого игрока $p \in P$, а min берется по всем чистым стратегиям второго $j = 1, ..., n$;
С помощью вспомогательной переменной u запишем задачу нахождения значения игры как:

$$v = \max (u),$$

max берется по всем $(u, p) \in B$, где

$$B = \{(u, p) \mid (p[1] \times a[1, j] + ... + p[m] \times a[m, j]) \geq u, j = 1..n; \\ p[1] + ... + p[m] = 1; p[i] \geq 0, i = 1..m \}$$

Таким образом при фиксированном $p \in P$ максимальное значение u:

$$\min (A(p, j)),$$

min берется по $1 \leq j \leq n$ - всем чистым стратегиям второго игрока.
Так как $v > 0$, будем считать u положительным. Сделаем замену переменных:

$$z[i] = p[i] / u$$

При такой замене:

$$z[1] + ... + z[m] = 1 / u;$$
$$a[1, j] \times z[1] + ... + a[m, j] \times z[m] \geq 1, j = 1..n;$$
$$z[i] \geq 0, i = 1..m;$$

Отсюда

$$v = 1 / (z`[1] + ... + z`[m]),$$

где $z` = (z`[1], ..., z`[m])$ - оптимальное решение задачи ЛП:

$$z[1] + ... + z[m] \rightarrow \min;$$
$$a[1, j] \times z[1] + ... + a[m, j] \times z[m] \geq 1, j = 1..n;$$
$$z[i] \geq 0, i = 1..m;$$

Отсюда находим значение игры v и оптимальную смешанную стратегию p` первого игрока по формулам:

$$v = 1 / (z`[1] + ... + z`[m]),$$
$$p` = v \times z`$$

Аналогично получаем вторую задачу ЛП из:

$$v = \min (\max (A(i, q))),$$

где min берется по всем смешанным стратегиям второго игрока $q \in Q$, а max берется по всем чистым стратегиям первого $i = 1, ..., m$;

$$v = 1 / (w`[1] + ... + w`[n]),$$

где $w` = (w`[1], ..., w`[n])$ - оптимальное решение задачи ЛП:

$$w[1] + ... + w[n] \rightarrow \max$$
$$a[i, 1] \times w[1] + ... + a[i, n] \times w[n] \leq 1, i = 1..m;$$
$$w[j] \geq 0, j = 1.. n;$$

Оптимальная смешанная стратегия второго игрока находится по формуле

$$q` = v \times w`;$$

Две описанные задачи ЛП двойственны по отношению друг к другу, так-как применяя правило перехода от прямой задачи к двойственной, можно получить одну из этих задач из другой и наоборот.
Правила перехода от прямой задачи к двойственной:
1. Вместо задачи на максимум, рассматривается задача на минимум
2. В ограничениях знаки ≥ меняем на ≤
3. Матрица ограничений A транспонируется
4. Коэффициенты правой части ограничений двойственной задачи являются коэффициентами целевой функции прямой задачи, и наоборот

Двойственная задача к двойственной задаче является прямой задачей

Реализация

Для реализации задачи были использованы следующие библиотеки:
- **scipy.optimize** – библиотека, которая содержит функцию **linprog**()
- **fractions** – содержит функцию **Fraction**() которая позволяет работать с обычными дробями
- **matplotlib.pyplot** – бибилотека для построения графиков, plt – псевдони
- **pylab** – вспомогательная библиотека для построения графиков

Основная функция, реализующая метод двойственных задач ЛП - **nash_equilibrium()**. Сначала опишем вспомогательные функции:

multiply_matrix(matrix, num) — функция умножения матрицы (одномерного списка, либо списка из списков) matrix на число num. Задаем цикл по длине матрицы, в нем проверяем одномерная это матрица или двумерная с помощью
if type(matrix[0]) == list
Если двумерная (список списков), то в двойном цикле умножаем каждый элемент на число, иначе просто умножаем каждый элемент списка на число.

transpose_matrix(matrix) — функция транспонирования матрицы matrix. Создаем новую матрицу new_matrix, в двойном цикле строим ее на основе матрицы-параметра, так чтобы выполнялось new_matrix[i][j] == matrix[j][i]. i = 1, …, m; j = 1, …, n.

print_matrix(matrix) — функция вывода матрицы matrix на экран. Аналогично функции умножения матриц пробегаем по длине matrix, проверяем массив ли это или матрица, соответствующе выводим элементы, разделяя их символом «|»

graph_str(strategy, player)- функция построения и вывода на экран графика спектра оптимальной стратегии strategy (список) игрока player (номер игрока). Создаем список x — список номеров стратегий игрока.
pylab.xlim([0, len(strategy) + 1])
pylab.ylim([0, max(strategy) * 1.25])

С помощью pylab.xlim, pylab.ylim, которые берут аргументы в виде списка из двух элементов — начального и конечного значения на оси, задаем длины осей x и y в единицах соответсвенно. Прибавляем 1 и умножаем на 1.25 в первом и втором случае соответственно для удобства представления графика.

plt.stem(x, strategy, '--') (обозначаем библиотеку matplotlib.pyplot как ptl(псевдоним)) — функция берет как первые аргументы списки значений на осях x и y соответственно, третий аргумент — строка, задающая вид вертикальных линий от точек, задаваемых координатами из первых двух аргументов, в данном случае — пунктирная линия.
plt.title(«») - принимает как аргумент строку, которая будет выведена заголовком над графиком. Выводим соответствующие заголовки для графиков первого, либо второго игрока в зависимости от номера player.
plt.show() - выводит график на экран в соответствии с заданными заранее данными.

print_solution(cost, str1, str2) — функция вывода ответа с визуализацией спектров оптимальных стратегий. Аргументы — значение игры (число) и оптимальные стратегии первого и второго игрока (списки). Используем вспомогательные функции вывода матриц и графиков, описанные ранее.

read_matrix(file) — считывание матрицы, задающей матричную игру из файла. Аргумент — имя файла. Создаем новый список mas, открываем файл на чтение, в цикле считываем матрицу, добавляя элементы в mas, закрываем файл, возвращаем mas.

fraction_matrix(matrix) — функция представления элементов матрицы matrix в виде рациональных дробей с числителем и знаменателем. Используется для удобства представления.
Fraction(n).limit_denominator() принимает число n и возвращает дробь, эквивалентную ближайшему к n рациональному числу (либо эквивалентную ему самому, если n рациональное).
В функции создаем новую матрицу и в цикле создаем ее на основе матрицы-параметра. Возвращаем созданную матрицу.

nash_equilibrium(matrix) — основная функция, принимает на вход матрицу, задающую матричную игру.
Сначала обозначаем m — количество стратегий первого игрока — кол-во строк матрицы, n - количество стратегий второго игрока — кол-во столбцов матрицы.

Далее реализуем сведение матричной игры к паре двойственных задач ЛП, решаем их с помощью функции scipy.optimize.linprog(c, a, b), реализующей симлекс-метод.
Функция linprog(c, a, b) принимает в качестве аргументов:
c — список коэффициентов целевой функции, которую надо минимизировать.
a- матрица коэффициентов при переменных в линейных ограничениях.
b- список верхних ограничений в линейных ограничениях задачи.
Для функции linprog условия неотрицательности значений переменных задаются по умолчанию.
Таким образом задаем и решаем пару двойственных задач ЛП. Видим, что параметр c первой задачи совпадает с параметром b второй, и наоборот. Также матрица a первой задачи — транспонированныя матрица из второй задачи.
Результат работы linprog(c, a, b) состоит из полей, в том числе включающих в себя массив значений переменных, на которых целевая функция достигает минимального значения (если таковые нашлись) и статус завершения.

Далее если оба вызова linprog завершились удачно
(проверка if first_res.success and second_res.success), то создаем списки из массивов значений переменных first_str и second_str, иначе выводим «Решений нет».

Далее находим значение игры по одной из двух формул:

$$v1 = 1 / (z[1] + \ldots + z[m])$$

$$v2 = 1 / (w[1] + \ldots + w[n]),$$

где [z[1], …, z[m] — список-решение первой двойственной задачи,
[w[1], …, w[n] — второй.

Далее списки first_str и second_str умножаются на получившееся значение игры, при этом получаем списки — оптимальные стратегии первого и второго игрока.

Значение и элементы списков сразу представляем в виде дробей с помощью функции Fraction(n).limit_denominator(), описанной ранее.

Возвращаем значение игры(число) и оптимальные стратегии игроков (списки).

Выполнение

Работу выполнили:

Артём Тангаев (311) - написание программы,
Екатерина Ворончихина (312) - написание readme